UNIT II

Relational Data Model

Data is represented in the relational model is in the form of **relation**.

A relation consists of a **relation schema** and a **relation instance**

- Relation instance is a table,
- Relation schema describes the column heads for the table.

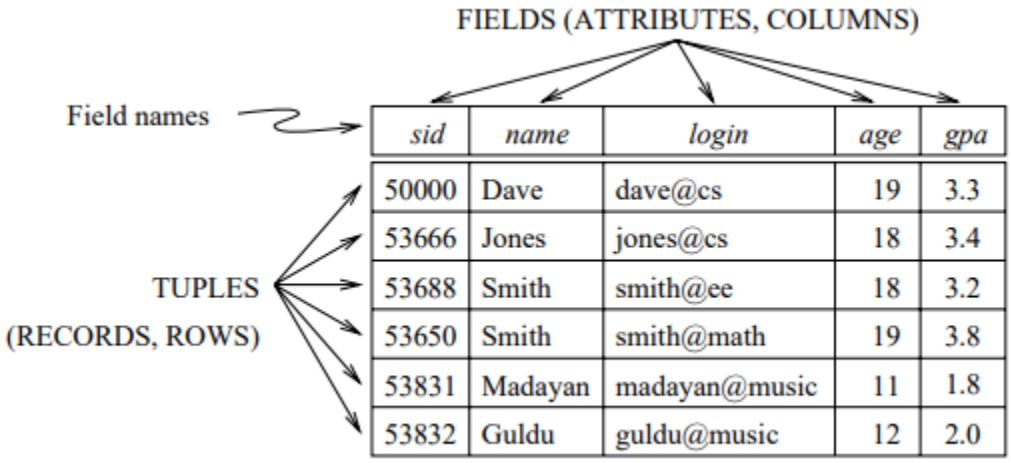
So, we first describe the relation schema and then the relation instance.

The schema specifies the relation's name, the name of each fiel**d** (or **column**, or **attribute**), and the **domain** of each fild. A domain is referred to in a relation schema by the **domain name** and has a set of associated **values**.

We use the example of student

- Students *(sid: string, name: string, login: string, age: integer, gpa: real)*

An **instance** of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema. A **relation instance** can be thought of as a table in which each tuple is a row, and all rows have the same number of fields.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

A relation schema specifies the domain of each field in the relation instance which specify an important condition that we want each instance of the relation to satisfy: The values that appear in a column must be drawn from the domain associated with that column.

Formally, let R(f1:D1, ..., fn:Dn) be a relation schema, and for each fi, $1 \leq i \leq n$, let Domi be the set of values associated with the domain named Di. An instance of R that satisfies the domain constraints in the schema is a set of tuples with n fields

{ hf1 : d1, ... ,fn : dni | d1 ∈ Dom1, ... ,dn ∈ Domn }

The degree, also called arity, of a relation is the number of fields. The cardinality of a relation instance is the number of tuples in it.

Creating and Modifying Relations

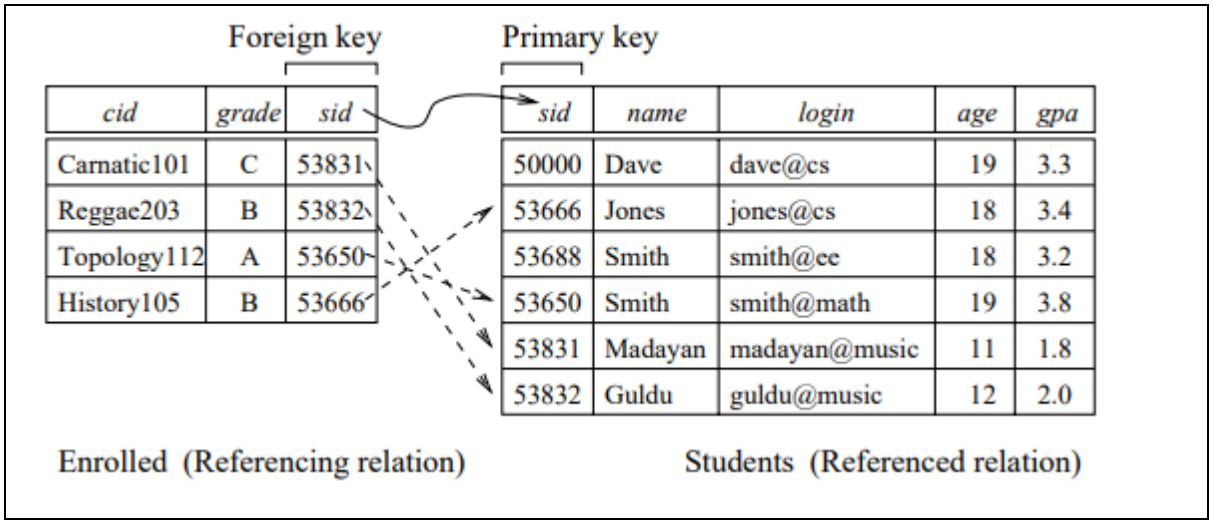
SQL-92 language standard uses the word table to denote relation.

Data Definition Language (DDL) is a subset of SQL that supports the creation, deletion, and modification of tables:

To create a table	**To insert values into table**
CREATE TABLE statement	INSERT statement
CREATE TABLE Students (sid CHAR(20), name CHAR(30), login CHAR(20), age INTEGER, gpa REAL)	INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)

To update records in a table	**To delete records**
UPDATE statement	DELETE statement
UPDATE Students S SET S.age = S.age + 1, S.gpa = S.gpa - 1 WHERE S.sid = 53688	DELETE FROM Students S WHERE S.name = 'Smith'

Integrity constraints (IC)	***Key Constraints***
It is the condition that is specified on a database schema, and restricts the data that can be stored in an instance of the database. If a database instance satisfies all the integrity constraints specified on the database schema, it is a legal instance. • When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database When a database application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs.	A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple A set of fields that uniquely identifies a tuple according to a key constraint is called a **candidate** key for the relation **Superkey**, is a set of fields that contains a key ***Example***: *CREATE TABLE Students (sid CHAR(20), name CHAR(30), login CHAR(20), age INTEGER, gpa REAL, UNIQUE (name, age), CONSTRAINT StudentsKey PRIMARY KEY (sid))*
Primary Key is a key that helps in uniquely identifying the tuple of the database **The foreign key** in the referencing relation must match the primary key of the referenced relation	



Querying database

A relational database query (query, for short) is a question about the data, and the answer consists of a new relation containing the result.

Example: SELECT * FROM Students S WHERE S.age < 18

SELECT S.name, S.login FROM Students S WHERE S.age < 18

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and produces instances of relations as output. It uses operators to perform queries.

Example

- Sailors *(sid: integer, sname: string, rating: integer, age: real)*
- Boats *(bid: integer, bname: string, color: string)*
- Reserves *(sid: integer, bid: integer, day: date)*

Basic operations:

- Selection (σ) Selects a subset of rows from relation.
- Projection (π) Selects a subset of columns from relation.
- Cross-product (×) Allows us to combine two relations.
- Set-difference ( ) Tuples in relation. 1, but not in relationn. 2.
- Union (U) Tuples in reln. 1 and in reln. 2.
- Rename(ρ) Use new name for the Tables or fields.

Additional operations:

Intersection (∩), Join( ), Division(÷): Not essential, but very useful. Since each operation returns a relation, operations can be composed! (Algebra is “closed”.)

Selection and Projection

Relational algebra includes operators to select rows from a relation (σ) and to project columns (π). These operations allow us to manipulate data in a single relation.

$\sigma_{rating>8}(S2)$

The selection operator σ specifies the tuples to retain through a selection condition. In general, the selection condition is a boolean combination of terms with attributes

projection operator π allows us to extract columns from a relation. It eliminates duplicates

For example, find out all sailor names and ratings by

$\pi_{sname,rating}(S2)$

Set Operations

Union

R∪S returns a relation instance containing all tuples that occur in either relation instance R or relation instance S (or both).

R and S must be union –compatible means that both have same number of fields and fields have same domains

Intersection:

R∩S returns a relation instance containing all tuples that occur in both R and S. The relations R and S must be union-compatible

Set-difference

R−S returns a relation instance containing all tuples that occur in R but not in S.

Cross-product

R ×S returns a relation instance whose schema contains all the fields of R (in the same order as they appear in R) followed by all the fields of S (in the same order as they appear in S).

The result of R × S contains one tuple hr, si (the concatenation of tuples r and s) for each pair of tuples r ∈ R, s ∈ S.

Renaming

The expression ρ(R(F), E) takes an arbitrary relational algebra expression E and returns an instance of a (new) relation called R. R contains the same tuples as the result of E, and has the same schema as E, but some fields are renamed.

Suppose,

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

Figure 4.1 Instance $S1$ of Sailors

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Figure 4.2 Instance $S2$ of Sailors

sid	*bid*	*day*
22	101	10/10/96
58	103	11/12/96

Figure 4.3 Instance $R1$ of Reserves

Then, the renaming ρ(C(1 → sid1, 5 → sid2), S1 × R1) would be

(sid)	*sname*	*rating*	*age*	*(sid)*	*bid*	*day*
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Joins

join operation combine information from two or more relations. It can be defined as a cross-product followed by selections and projections.

Conditional join

This join operation accepts a join condition c and a pair of relation instances as arguments, and returns a relation instance.

$R \bowtie_c S = \sigma c(R \times S)$

Note that the condition c can refer to attributes of both R and S.

Equijoin

Join operation R⋈ S is when the join condition consists solely of equalities (connected by ∧) of the form R.name1 = S.name2. Additional projection in which S.name2 is dropped when the result is returned.

sid	*sname*	*rating*	*age*	*bid*	*day*
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Figure 4.13 $S1 \bowtie_{R.sid=S.sid} R1$

Natural Join

A special case of the join operation R⋈ S is an equijoin in which equalities are specified on all fields having the same name in R and S. It has the nice property that the result is guaranteed not to have two fields with the same name.

Division

Consider two relation instances A and B in which A has (exactly) two fields x and y and B has just one field y. We define the division operation A/B as the set of all x values (in the form of unary tuples) such that for every y value in (a tuple of) B, there is a tuple hx,yi in A..

A

sno	*pno*
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

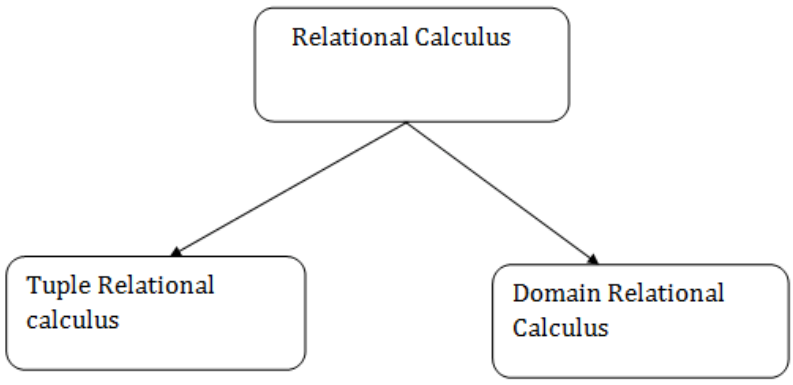
sno
s1
s4

A/B3

sno
s1

Relational Calculus

- Relational calculus is an alternative to relational algebra.
- In contrast to the algebra, which is procedural, the calculus is nonprocedural, or declarative,



Tuple Relational Calculus

The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.The result of the relation can have one or more tuples.

Notation:

{T | P (T)} or

{T | Condition (T)}

Where **T** is the resulting tuples and **P(T)** is the condition used to fetch T.

For example, to find all students whose age is greater than 22

{ t.Last_Name | Student(t) AND t.age > 30 }

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

Another example,

Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

{t| t ∈ loan ∧ t[amount]>=10000}

Formal Definition

Let Rel be a relation name, R and S be tuple variables, a an attribute of R, and b an attribute of S. Let op denote an operator in the set {, =, ≤, ≥, 6=}. An atomic formula is one of the following:

- R ∈ Rel
- R.a op S.b
- R.a op constant, or constant op R.a

A formula is recursively defined to be one of the following, where p and q are themselves formulas, and p(R) denotes a formula in which the variable R appears:

any atomic formula

- ¬p, p ∧ q, p ∨ q, or p ⇒ q
- ∃R(p(R)), where R is a tuple variable
- ∀R(p(R)), where R is a tuple variabl

Domain Relational Calculus

In domain relational calculus the records are filtered based on the domains. A domain variable is a variable that ranges over the values in the domain of some attribute. In domain relational calculus, filtering variable uses the domain of attributes. It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

A DRC query has the form

{hx1, x2,...,xni | p(hx1, x2,...,xni)},

where each xi is either a domain variable or a constant and p(hx1, x2,...,xni) denotes a DRC formula whose only free variables are the variables among the xi, 1 ≤ i ≤ n. The result of this query is the set of all tuples hx1, x2,...,xni for which the formula evaluates to true

An atomic formula in DRC is one of the following:

- hx1, x2,...,xni ∈ Rel, where Rel is a relation with n attributes; each xi, 1 ≤ i ≤ n is either a variable or a constant.
- X op Y
- X op constant, or constant op X

A formula is recursively defined to be one of the following, where p and q are themselves formulas, and p(X) denotes a formula in which the variable X appears:

- any atomic formula
- ¬p, p ∧ q, p ∨ q, or p ⇒ q
- ∃X(p(X)), where X is a domain variable
- ∀X(p(X)), where X is a domain variabl

For example,

To find the first name and age of students where student age is greater than 27,

{ < First_Name, Age > | ∈Student ∧ Age > 27 }

SQL

IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory

Renamed Structured Query Language (SQL).

The SELECT statement is used to select data from a database.

Basic structure of an SQL expression consists of select, from and where clauses.

- Select clause lists attributes to be copied - corresponds to relational algebra project.
- from clause corresponds to Cartesian product - lists relations to be used.
- where clause corresponds to selection predicate in relational algebra

The data returned is stored in a result table, called the result-set. To fetch the entire table or all the fields in the table:

SELECT * FROM table_name;

To fetch individual column data

SELECT column1,column2 FROM table_name WHERE SQL claus

WHERE clause is used to specify/apply any condition while retrieving, updating or deleting data from a table

For example, fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 20000

SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 20000

From clause: From clause can be used to specify a sub-query expression in SQL. The relation produced by the sub-query is then used as a new relation on which the outer query is applied.

- Sub queries in the from clause are supported by most of the SQL implementations.
- The correlation variables from the relations in from clause cannot be used in the subqueries in the from clause.

Syntax:

SELECT column1, column2 FROM (SELECT column_x as C1, column_y FROM table WHERE PREDICATE_X) as table2 WHERE PREDICATE;

SET Operations

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions. Wel cover 4 different types of SET operations, along with example:

- UNION
- UNION ALL
- INTERSECT
- MINUS

Union

SQL Union operation is used to combine the result of two or more SQL SELECT queries. In the union operation, all the columns and its data type must be same in both the tables on which UNION operation is being applied.

The union operation eliminates the duplicate rows from its resultset

Syntax

SELECT column_name FROM table1 UNION SELECT column_name FROM table2;

Union All

Union All operation is equal to the Union operation.

It returns the set without removing duplication and sorting the data.

Syntax:

SELECT column_name FROM table1 UNION ALL SELECT column_name FROM table2;

Intersect

It combines two SELECT statement sand returns the common rows from both the SELECT statements. Columns and its data types must be the same in both tables.

It has no duplicates and it arranges the data in ascending order by default

Syntax

SELECT column_name FROM table1 INTERSECT SELECT column_name FROM table2;

Minus

It combines the result of two SELECT statements which display the rows which are present in the first query but absent in the second query.

It has no duplicates and data arranged in ascending order by default.

Syntax:

SELECT column_name FROM table1 MINUS SELECT column_name FROM table2;

Aggregate functions in SQL

They are used to perform the calculations on multiple rows of a single column of a table.

It returns a single value like summarize the data.

- Count() - Count the number of rows in a database table, Count(*) returns total number of records
- Sum() - calculate the sum of all selected columns; syntax is SUM([ALL|DISTINCT] expression)
- Avg() - calculate the average value of the numeric type
- Min() - find the maximum value of a certain column
- Max() - find the minimum value of a certain column

GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) ORDER BY column_name(s);

HAVING Clause:

We use HAVING clause to place conditions to decide which group will be the part of final result-set. We can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause

Syntax:

SELECT column1, function_name(column2) FROM table_name WHERE condition GROUP BY column1, column2 HAVING condition ORDER BY column1, column2;

Nested Query

A nested query is a query that has another query embedded within it; the embedded query is called a subquery. The query used to compute this subsidiary table is a subquery and appears as part of the main query. A subquery typically appears within the WHERE clause of a query.

For example, find the names of sailors who have reserved boat 103

SELECT S.sname FROM Sailors S WHERE S.sid IN (SELECT R.sid FROM Reserves R WHERE R.bid = 103

Views

Views in SQL are considered as a virtual table that contains both rows and columns. To create the view, we select the all/ specific fields from one or more tables present in the database. It can be used as abstraction layer between user and table and created by SELECT statement.

A view can either have specific rows based on certain condition or all the rows of a table

Create a view by

Syntax: CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE condition;

Example -1 (for single table)

create view All-customer as (select branch-name, customer-name from Depositor, Account where Depositor.account-number=account.account-number)

Example -2 (for multiple tables)

create view All-customer as (select branch-name, customer-name from Depositor, Account where Depositor.account-number=account.account-number)

union

(select branch-name, customer-name from Borrower, Loan where Borrower.loan-number = Loan.loan-number)

View can be deleted by

DROP VIEW <view name>

View can be deleted by

CREATE OR REPLACE VIEW view_name AS SELECT column1, column2, ... FROM table_name WHERE condition;

WITH CHECK OPTION is a CREATE VIEW statement option used to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition. If they do not satisfy the condition, the UPDATE or INSERT returns an error.

CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age
FROM CUSTOMERS WHERE age IS NOT NULL WITH CHECK OPTION;

It doesn't allow null value in the age column

Advantages of views

- It help to reduce the complexity by allowing to create different views on the same base table for different users.

- increases the security by excluding the sensitive information from the view
- it is Consistent, view is a unchanged image of the structure of the database
- If data is accessed and entered through a view, it meets the specified integrity constraints
- Views take very little space to store the data
- View makes application and database to a certain extent independent

Disadvantages

- can't pass parameters to the SQL server views
- can't associate rules and defaults with views
- can't be created view on temporary tables

Types of view

Simple view	***Complex view***	***Inline view***	***Materialized view***
It is based on single table and not contain GROUP BY clause or functions	It is based on multiple tables which contain GROUP BY clause or functions	It is based on sub query in FROM clause that creates temporary table	It stores definition as well as data. It creates replicas of data

Constraints

Constraints are restrictions like what values are allowed to be inserted in the relation, what kind of modifications and deletions are allowed in the relation.

Categories of Constraints

Constraints are applied in the data model is called **Implicit constraints**

Constraints that are directly applied in the schemas of the data model, by DDL (Data Definition Language), called as schema-based or **Explicit constraints**

Constraints that cannot be directly applied in the schemas of the data model, called as Application based or **semantic constraints**

Constraints type

Domain constraints

Every domain must contain atomic values (smallest indivisible units). It means that composite and multi-valued attributes are not allowed. We perform datatype check here, which means when we assign a data type to a column we limit the values that it can contain.

For example, If we assign the datatype of attribute age as int, we cant give it values other then int datatype.

Key constraints

These are called uniqueness constraints since it ensures that every tuple in the relation should be unique. A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes. Null values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.

Entity constraints

Entity Integrity constraints says that no primary key can take NULL value, since using primary key we identify each tuple uniquely in a relation

Referential Integrity Constraints

The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations. This constraint is enforced through foreign key. When an attribute in the foreign key of relation R1 have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.

The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.